UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/684,662	10/14/2003	Douglas R. Armstrong	20002/15251	3392

75343 7590 11/12/2008
Hanley, Flight & Zimmerman, LLC
150 S. Wacker Drive
Suite 2100
Chicago, IL 60606

EXAMINER	
WANG, BEN C	
ART UNIT	**PAPER NUMBER**
2192	
MAIL DATE	**DELIVERY MODE**
11/12/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

PTOL-90A (Rev. 04/07)

Office Action Summary	**Application No.** 10/684,662	**Applicant(s)** ARMSTRONG ET AL.
	Examiner BEN C. WANG	**Art Unit** 2192

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *25 January 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-19 and 21-42* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-19 and 21-42* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.	6) ☐ Other: _____.

DETAILED ACTION

1. Applicant's amendment dated January 25, 2008, responding to the Office Action mailed July 25, 2007 provided in the rejection of claims 1-19 and 21-42.

Claims 1-19 and 21-42 remain pending in the application and which have been fully considered by the examiner.

2. Examiner notices that the applicant has signed date of "January 25, 2007" of the submitted Amendment on pages 1 and 11 respectively, which should be corrected as January 25, 2008.

3. Applicant's arguments with respect to claims currently amended have been fully considered but are not persuasive. Please see the section of "Response to Arguments" for details.

4. Accordingly, **THIS ACTION IS MADE FINAL.** See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR

1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

5. Claims 1-2, 11-12, 21-22, 32-33, and 42 are rejected under 35 U.S.C. 103(a) as being unpatentable over Robert J. Hall (*CPPROFJ: Aspect-capable Call Path Profiling of Multi-threaded Java Applications, 2002, IEEE, pp. 1-10*) (hereinafter 'Hall') in view of Jeffrey K. Hollingsworth (*Critical Path Profiling of Message Passing and Shared-Memory Program, Jan. 1998, IEEE, pp. 1029-1040*) (hereinafter 'Hollingsworth-2')

6. **As to claim 1** (Original), Hall discloses a method of profiling a threaded program during program runtime, the method comprising:

- monitoring information exchanged between a processing unit and first and second threads executed by the processing unit (e.g., Sec. 1 – Introduction, 1st

Bullet - ... call path profiling to handle multi-threaded server-style Java applications through the definition of two views: the Thread View, which profiles costs incurred by instances of single thread class, and the Server View, which **profiles CPU costs of all threads** ...; 3rd Bullet - .. The **Thread View profiles** report (a) time spent when actually running, (b) time spent sharing **the CPU** with **equal priority threads**, (c) time spent waiting while **higher priority threads** are **running**, (d) time spent waiting for a monitor, and (e) idle time waiting for other **types of events** ...; Abstract, Lines 10-21 - first, it provides two different call path oriented views on program performance, a server view and a thread view; the former helps one optimize for throughput, while the latter is useful for optimizing thread latency; the views incorporate a typed time notation for representing different program activities, such as monitor wait and thread preemption times; second, the new framework allows aspect-oriented program profiling, even when the original program was not designed in an aspect oriented fashion; finally, the approach is implemented in a too, CPPROFJ, an aspect-capable call path profile for Java™);

- determining, based on the information exchanged between the processing unit and the first and second threads, a wait time during which the first thread awaits a synchronization event (e.g., Sec. 1- Introduction, 4th Para - a third source of performance bottlenecks in modern applications is thread contention and other inter-thread dependencies; for example, an input/output processing thread may b stalled waiting for a thread marked with higher priority, event thought its data has arrived from the input device; it can often be better to mark the 110 processor as higher priority so that it can start the next read or write before letting the more compute-bound threads continue); and

- determining whether the wait time affects the critical path of thread execution (e.g., Sec. 1 -Introduction, 5th Para, 3rd bullet -the thread view profiles report (a) time spent when actually running, (b) time spent sharing the CPU with equal priority threads, (c) time spent waiting while higher priority threads are running, (d) time spent waiting for a monitor, and (e) idle time waiting for other types of events)

Hall does not explicitly disclose determining, based on the information exchanged between the processing unit and the first and second threads, a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree.

However, in an analogous art of *Critical Path Profiling of Message Passing and Shared-Memory Program,* Hollingsworth-2 discloses determining, based on the information exchanged between the processing unit and the first and second threads, a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree (e.g., Fig. 1 - a program activity graph and calculating it critical path; Sec. 2 - Critical Path; Sec. 2.1 -Formal Definition of Critical Path, 1st and 2nd Para - ... The **critical path** of a parallel program is **the longest path through the PAG (Program Activity Graph)** ...; Fig. 1- A program activity graph and calculating its critical path; Sec. 1- Critical Path, 1st Para - The diagonal arcs show the communication events between processes and the dashed line **<u>shows the critical path</u>** through the program's execution ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hollingswarth-2 into the Hall's system to further provide determining, based on the information exchanged between the processing unit and the first and second threads, a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree in Hall system.

The motivation is that it would further enhance the Hall's system by taking, advancing and/or incorporating Hollingsworth-2's system which offers significant advantages of critical path profiling compared to metrics that simply add values for individual processes is that is provides a "global view" of the performance of a parallel computation that captures performance implications of the interactions between processes; and, in a distributed system, extracting a global view during the computation requires exchanging information between processes as once suggested by Hollingsworth-2 (e.g., Sec. 2 - Critical Path, 2nd Para)

7. **As to claim 2** (Original) (incorporating the rejection in claim I), Hall discloses indicating that the wait time is of a high priority if the wait time affects the critical path of thread execution and indicating that the wait time is of a low priority if the wait time does not affect the critical path of thread execution (e.g., Sec. I-Introduction, 5th Para, 3rd

bullet - the thread view profiles report (a) time spent when actually running, (b) time spent sharing the CPU with equal priority threads, (c) time spent waiting while higher priority threads are running, (d) time spent waiting for a monitor, and (e) idle time waiting for other types of events)

8. **As to claim 11** (Original), Hall discloses an article of manufacture comprising a machine-accessible medium having a plurality of machine-accessible instructions that, when executed, causes a machine to:

- monitor information exchanged between a processing unit and first and second threads executed by the processing unit (e.g., Sec. 1 – Introduction, 1st Bullet - … call path <u>profiling to handle</u> <u>multi-threaded server-style Java applications</u> through the definition of two views: the Thread View, which profiles costs incurred by instances of single thread class, and the Server View, which <u>**profiles CPU costs of all threads**</u> …; 3rd Bullet - .. <u>The **Thread View** **profiles** report</u> (a) time spent when actually running, (b) <u>time spent sharing **the CPU**</u> <u>with **equal priority threads**</u>, (c) <u>time spent waiting while **higher priority threads** are **running**</u>, (d) time spent waiting for a monitor, and (e) <u>idle time waiting for other **types of events**</u> …; Abstract, Lines 10-21 - first, it provides two different call path oriented views on program performance, a server view and a thread view; the former helps one optimize for throughput, while the latter is useful for optimizing thread latency; the views incorporate a typed time notation for representing different program activities, such as monitor wait and thread preemption times; second, the new framework allows aspect-oriented program profiling, even when the original program was not designed in an aspect oriented fashion; finally, the approach is implemented in a too, CPPROFJ, an aspect-capable call path profile for Java™);

- determine, based on the information exchanged between the processing unit and the first and second threads, a wait time during which the first thread awaits a synchronization event (e.g., Sec. 1- Introduction, 4th Para - a third source of performance bottlenecks in modern applications is thread contention and other inter-thread dependencies; for example, an input/output processing thread may b stalled waiting for a thread marked with higher priority, event thought its data has arrived from the input device; it can often be better to mark the 110 processor as higher priority so that it can start the next read or write before letting the more compute-bound threads continue); and
- determine whether the wait time affects the critical path of thread execution (e.g., Sec. 1-Introduction, 5th Para, 3rd bullet -the thread view profiles report (a) time spent when actually running, (b) time spent sharing the CPU with equal priority threads, (c) time spent waiting while higher priority threads are running, (d) time spent waiting for a monitor, and (e) idle time waiting for other types of events).

Hall does not explicitly disclose determining, based on the information exchanged between the processing unit and the first and second threads, a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree.

However, in an analogous art of *Critical Path Profiling of Message Passing and Shared-Memory Program*, Hollingsworth-2 discloses determining, based on the information exchanged between the processing unit and the first and second threads, a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree (e.g., Fig. 1 - a program activity graph and calculating it critical path; Sec. 2 - Critical Path; Sec. 2.1 -Formal Definition of Critical Path, 1st and 2nd Para - ... The **critical path** of a parallel program is **the longest path through the PAG (Program Activity Graph)** ...; Fig. 1- A program activity graph and calculating its critical path; Sec. 1- Critical Path, 1st Para - The diagonal arcs show the communication events between processes and the dashed line **shows the critical path** through the program's execution ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hollingsworth-2 into the Hall's system to further provide determining, based on the information exchanged between the processing unit and the first and second threads, a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree in Hall system.

The motivation is that it would further enhance the Hall's system by taking, advancing and/or incorporating Hollingsworth-2's system which offers significant advantages of critical path profiling compared to metrics that simply add values for individual processes is that is provides a "global view" of the performance of a parallel computation that captures performance implications of the interactions between processes; and, in a distributed system, extracting a global view during the computation requires exchanging information between processes as once suggested by Hollingsworth-2 (e.g., Sec. 2 - Critical Path, 2nd Para)

9. **As to claim 12** (Original) (incorporating the rejection in claim 11), please refer to claim **2** above, accordingly

10. **As to claim 21** (Original), Hall discloses a method of profiling a threaded program during program runtime, the method comprising:

- monitoring information exchanged between a processing unit and first and second threads executed by the processing unit (e.g., Sec. 1 – Introduction, 1st Bullet - … call path <u>profiling to handle</u> <u>multi-threaded server-style Java applications</u> through the definition of two views: the Thread View, which profiles costs incurred by instances of single thread class, and the Server View, which <u>**profiles CPU costs of all threads**</u> …; 3rd Bullet - .. <u>The **Thread View profiles** report</u> (a) time spent when actually running, (b) <u>time spent sharing **the CPU**</u> <u>with **equal priority threads**</u>, (c) <u>time spent waiting while **higher priority threads** are **running**</u>, (d) time spent waiting for a monitor, and (e) <u>idle time waiting for other **types of events**</u> …; Abstract, Lines 10-21 - first, it provides two different call path oriented views on program performance, a server view and a thread view; the former helps one optimize for throughput, while the latter is useful for optimizing thread latency; the views incorporate a typed time notation for representing different program activities, such as monitor wait and thread preemption times; second, the new framework allows aspect-oriented program profiling, even when the original program was not designed in an aspect oriented fashion; finally, the approach is implemented in a too, CPPROFJ, an aspect-capable call path profile for Java™);

- determining, based on the cross-thread event and the information exchanged between the processing unit and the first and second threads (e.g., Sec. 1 - Introduction,4' Para -a third source of performance bottlenecks in modern applications is thread contention and other inter-thread dependencies; for example, an input/output processing thread may b stalled waiting for a thread marked with higher priority, event thought its data has arrived from the input device; it can often be better to mark the 110 processor as higher priority so that it can .start the next read or write before letting the more compute-bound threads continue), a wait time during which the first thread awaits a synchronization event; and determining whether the wait time affects the critical path of thread execution (e.g., Sec. 1- Introduction, 5th Para, 3rd bullet - the thread view profiles report (a) time spent when actually running, (b) time spent sharing the CPU with equal priority threads, (c) time spent waiting while higher priority threads are running, (d) time spent waiting for a monitor, and (e) idle time waiting for other types of events).

Hall does not explicitly disclose determining, based on the cross-thread event, a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree.

However, in an analogous art of *Critical Path Profiling of Message Passing and Shared-Memory Program*, Hollingsworth-2 discloses determining, based on the cross-thread event, a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree (e.g., Fig. 1 - a program activity graph and calculating it critical path; Sec. 2 - Critical Path; Sec. 2.1 -Formal Definition of Critical Path, 1st and 2nd Para - ... The **critical path** of a parallel program is **the longest path through the PAG (Program Activity Graph)** ...; Fig. 1- A program activity graph and calculating its critical path; Sec. 1- Critical Path, 1st Para - The diagonal arcs show the communication events between processes and the dashed line **shows the critical path** through the program's execution ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hollingsworth-2 into the Hall's system to further provide determining, based on the cross-thread event, a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree in Hall system.

The motivation is that it would further enhance the Hall's system by taking, advancing and/or incorporating Hollingsworth-2's system which offers significant advantages of critical path profiling compared to metrics that simply add values for individual processes is that is provides a "global view" of the performance of a parallel computation that captures performance implications of the interactions between processes; and, in a distributed system, extracting a global view during the computation requires exchanging information between processes as once suggested by Hollingsworth-2 (e.g., Sec. 2 - Critical Path, 2ndPara)

11. **As to** claim **22** (Original) (incorporating the rejection in claim 21) a, Hall discloses indicating that the wait time is of a high priority if the wait time affects the critical path of thread execution and indicating that the wait time is of a low priority if the wait time does not affect the critical path of thread execution (e.g., Sec. 1 - Introduction, 5thPara, **3rd**bullet -the thread view profiles report (a) time spent when actually running, (b) time spent sharing the CPU with equal priority threads, (c) time spent waiting while higher priority threads are running, (d) time spent waiting for a monitor, and (e) idle time waiting for other types of events)

12. **As to claim 32** (Original), Hall discloses an article of manufacture comprising a machine-accessible medium having a plurality of machine-accessible instructions, when executed, causes a machine to:

- monitor information exchanged between a processing unit and first and second threads executed by the processing unit (e.g., Sec. 1 – Introduction, 1st Bullet - ... call path <u>profiling to handle</u> <u>multi-threaded server-style Java applications</u> through the definition of two views: the Thread View, which profiles costs incurred by instances of single thread class, and the Server View, which **profiles CPU costs of all threads** ...; 3rd Bullet - .. The **Thread View profiles** report (a) time spent when actually running, (b) time spent sharing **the CPU** with **equal priority threads**, (c) time spent waiting while **higher priority threads** are **running**, (d) time spent waiting for a monitor, and (e) idle time waiting for other **types of events** ...; Abstract, Lines 10-21 - first, it provides two different call path oriented views on program performance, a server view and a thread view; the former helps one optimize for throughput, while the latter is useful for optimizing thread latency; the views incorporate a typed time notation for representing different program activities, such as monitor wait and thread preemption times; second, the new framework allows aspect-oriented program profiling, even when the original program was not designed in an aspect oriented fashion; finally, the approach is implemented in a too, CPPROFJ, an aspect-capable call path profile for Java™);

- determine, based on the cross-thread event and the information exchanged between the processing unit and the first and second threads (e.g., Sec. 1-Introduction, 4th Para -a third source of performance bottlenecks in modern applications is thread contention and other inter-thread dependencies; for example, an input/output processing thread may b stalled waiting for a thread marked with higher priority, event thought its data has arrived from the input device; it can often be better to mark the 110 processor as higher priority so that it can start the next read or write before letting the more compute-bound threads continue), a wait time during which the first thread awaits a synchronization event; and

- determine whether the wait time affects the critical path of thread execution (e.g., Sec. 1-Introduction, 5th Para, 3rd bullet -the thread view profiles report (a) time spent when actually running, (b) time spent sharing the CPU with equal priority threads, (c) time spent waiting while higher priority threads are running, (d) time spent waiting for a monitor, and (e) idle time waiting for other types of events).

Hall does not explicitly disclose determining, based on the cross-thread event, a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree.

However, in an analogous art of *Critical Path Profiling of Message Passing and Shared-Memory Program*, Hollingsworth-2 discloses determining, based on the cross-thread event, a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree (e.g., Fig. 1 - a program activity graph and calculating it critical path; Sec. 2 - Critical Path; Sec. 2.1 -Formal Definition of Critical Path, 1st and 2nd Para - … The **critical path** of a parallel program is **the longest path through the PAG (Program Activity Graph)** …; Fig. 1- A program activity graph and calculating its critical path; Sec. 1- Critical Path, 1st Para - The diagonal arcs show the communication events between processes and the dashed line **shows the critical path** through the program's execution …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hollingsworth-2 into the Hall's system to further provide determining, based on the cross-thread event, a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree in Hall system.

The motivation is that it would further enhance the Hall's system by taking, advancing and/or incorporating Hollingsworth-2's system which offers significant advantages of critical path profiling compared to metrics that simply add values for individual processes is that is provides a "global view" of the performance of a parallel computation that captures performance implications of the interactions between processes; and, in a distributed system, extracting a global view during the computation requires exchanging information between processes as once suggested by Hollingsworth-2 (e.g., Sec. 2 -Critical Path, 2"d Para)

13. **As to claim 33** (Original) (incorporating the rejection in claim 32), please refer to claim **22** above, accordingly

14. **As to** claim **42** (Previously Presented) (incorporating the rejection in claim I), Hollingsworth-2 discloses a method wherein the information includes one or more timestamps (e.g., Fig. 1- A program activity graph and calculating its critical path; Sec. 2 - Critical Path, 1st Para - a simple definition of the critical path of a program is

the longest, time-weighted sequence of events from the start of the program to its termination)

15. Claims 4-5, 7-9, 14-15, 17-19,25-26, 28-30, 35-36 ,and 38-40 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hall in view of Hollingsworth-2, and further in view of Broberg et al., *(Performance Optimization Using Critical Path Analysis in Multithreaded Programs on Multiprocessors, 1999,Psilander Grafiska, Karlskrona, Sweden, pp. 1-12)* (hereinafter 'Broberg')

16. **As to claim 4** (Original) (incorporating the rejection in claim I), Hall and Hollingsworth-2 do not explicitly disclose a leaf is added to the critical path tree when the synchronization event is a signal event.

However, in an analogous art of *Performance Optimization using Critical Path Analysis in Multithreaded Programs on Multiprocessors*, Broberg discloses a leaf is added to the critical path tree when the synchronization event is a signal event (e.g., Sec. 2 - Overview of the Critical Path Analysis, 2nd Para, 3rd Para; Sec. 3.1 -Finding the critical path in the optimal case, 3rd Para)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Broberg into the Hall-Hollingsworth-2's system to further provide a leaf is added to the critical path tree when the synchronization event is a signal event in Hall-Hollingsworth-2 system.

The motivation is that it would further enhance the Hall-Hollingsworth-2's. . system by taking, advancing and/or incorporating Broberg's system which offers significant advantages for finding the critical path of the multi-threaded program and the optimization is only done on those specific code segments of the program as once suggested by Broberg (e.g., Abstract)

17. **As to claim 5** (Original) (incorporating the rejection in claim I), Broberg discloses a leaf is removed from the critical path tree when the synchronization event is a wait event (e.g., Sec. 2 - Overview of the Critical Path Analysis, 2nd Para, 3rd Para; Sec. 3.1 - Finding the critical path in the optimal case, 3rd Para)

18. **As to claim 7** (Original) (incorporating the rejection in claim I), Broberg discloses a leaf is removed from the critical path tree when the synchronization event is a block event (e.g., Sec. 3.1 - Finding the critical path in the optimal case, 4th Para, 5thPara; Sec. 3.3 –Finding the critical path with CPU constraints, 3rd Para, 4th Para)

19. **As to claim 8** (Original) (incorporating the rejection in claim I), Broberg discloses a leaf is removed from the critical path tree when the synchronization event is a suspend event (e.g., Sec. 2 - Overview of the Critical Path Analysis, 2nd Para, 3rd Para; Sec. 3.1 - Finding the critical path in the optimal case, 3rd Para)

20. **As to claim 9** (Original) (incorporating the rejection in claim I), Broberg discloses a leaf is added to the critical path tree when the synchronization event is a resume event (e.g., Sec. 2 -Overview of the Critical Path Analysis, 2nd Para, 3rd Para; Sec. 3.1 – Finding the critical path in the optimal case, 3rd Para)

21. **As to claim 14** (Original) (incorporating the rejection in claim 12), please refer to claim **4** above, accordingly

22. **As to claim 15** (Original) (incorporating the rejection in claim 12), please refer to claim **5** above, accordingly

23. **As to claim 17** (Original) (incorporating the rejection in claim 12), please refer to claim **7** above, accordingly

24. **As to claim 18** (Original) (incorporating the rejection in claim 12), please refer to claim **8** above, accordingly

25. **As to claim 19** (Original) (incorporating the rejection in claim 12), please refer to claim **9** above, accordingly.

26. **As to claim 25** (Original) (incorporating the rejection in claim 22), Broberg discloses a leaf is added to the critical path tree when the synchronization event is a signal event (e.g., Sec. 2 - Overview of the Critical Path Analysis, 2nd Para, 3rd Para; Sec. 3.1 - Finding the critical path in the optimal case, 3rd Para)

27. **As to claim 26** (Original) (incorporating the rejection in claim 22), Broberg discloses a leaf is removed from the critical path tree when the synchronization event is a wait event (e.g., Sec. 2 -Overview of the Critical Path Analysis, 2nd Para, 3rd Para; Sec. 3.1 - Finding the critical path in the optimal case, 3rd Para)

28. **As to claim 28** (Original) (incorporating the rejection in claim 22), Broberg discloses a leaf is removed from the critical path tree when the synchronization event is a block event (e.g., Sec. 3.1 -Finding the critical path in the optimal case, 4th Para, 5th Para; Sec. 3.3 - Finding the critical path with CPU constraints, 3rd Para, 4th Para)

29. **As to claim 29** (Original) (incorporating the rejection in claim 22), Broberg discloses a leaf is removed from the critical path tree when the synchronization event is a suspend event (e.g., Sec. 2 -Overview of the Critical Path Analysis, 2nd Para, 3rd Para; Sec. 3.1 - Finding the critical path in the optimal case, 3rd Para)

30. **As to claim 30** (Original) (incorporating the rejection in claim 22), Broberg discloses a leaf is added to the critical path tree when the synchronization event is a resume event (e.g., Sec. 2 -Overview of the Critical Path Analysis, 2nd Para, 3rd Para; Sec. 3.1 - Finding the critical path in the optimal case, 3rd Para)

31. **As to claim 35** (Original) (incorporating the rejection in claim 32), please refer to claim **25** above, accordingly.

32. **As to claim 36** (Original) (incorporating the rejection in claim 32), please refer to claim **26** above, accordingly.

33. **As to claim** 38 (Original) (incorporating the rejection in claim 32), please refer to claim **28** above, accordingly.

34. **As to claim 39** (Original) (incorporating the rejection in claim 32), please refer to claim **29** above, accordingly.

35. **As to claim 40** (Original) (incorporating the rejection in claim 32), please refer to claim **30** above, accordingly.

36. Claims 3, 6, 10, 13, 16, 24, 27, 31, 34, 37, and 41 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hall in view of Hollingsworth-2, and further in view of Intel™ Technology Journal (Hyper-Threading Technology, Feb. 2002, Intel™ Technology Journal, pp. 1-66) (hereinafter 'Intel')

37. **As to claim 3** (Original) (incorporating the rejection in claim I), Hall and Hollingsworth-2 do not explicitly disclose a leaf is added to the critical path tree when the synchronization event is a fork event.

However, in an analogous art of *Performance Optimization using Critical Path Analysis in Multithreaded Programs on Multiprocessors*, Intel discloses a leaf is added to the critical path tree when the synchronization event is a fork event (e.g., P. 39, Right-Col. 1 Para)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Intel into the Hall-Hollingsworth-2's system to further provide a leaf is added to the critical path tree when the synchronization event is a fork event in Hall-Hollingsworth-2 system.

The motivation is that it would further enhance the Hall-Hollingsworth-2's system by taking, advancing and/or incorporating Intel's system which offers significant advantages that Intel's Hyper-Threading Technology delivers two logical processors that can execute different tasks simultaneously using shared hardware resources; and, speculative pre-computation, a technique that improves the latency of single-threaded applications by utilizing idle multithreading hardware resources to perform long-range data pre-fetches as once suggested by Intel (e.g., P. 2, 2nd Para, 3rd Para)

38. **As to** claim **6** (Original) (incorporating the rejection in claim l), Intel discloses a leaf is added to the critical path tree when the synchronization event is an entry event (e.g., P. 38, Sec. of Multi-Entry Threading)

39. **As to claim 10** (Original) (incorporating the rejection in claim l), Intel discloses comparing a number of active threads to a number of processing resources to determine a utilization factor (e.g., P. 2, 3rd Para; P.3, 2nd Para)

40. **As to claim 13** (Original) (incorporating the rejection in claim 12), please refer to claim **3** above, accordingly.

41. **As to** claim **16** (Original) (incorporating the rejection in claim 12), please refer to claim **6** above, accordingly.

42. **As to** claim **24** (Original) (incorporating the rejection in claim 22), Intel discloses a leaf is added to the critical path tree when the synchronization event is a fork event (e.g., P. 39, Right-Col, 1st Para)

43. **As to** claim **27** (Original) (incorporating the rejection in claim 22), Intel discloses a leaf is added to the critical path tree when the synchronization event is an entry event (e.g., P. 38, Sec. of Multi-Entry Threading)

44. **As to claim 31** (Original) (incorporating the rejection in claim 22), Intel discloses comparing a number of active threads to a number of processing resources to determine a utilization factor (e.g., P. 2, 3rd Para; P.3, 2nd Para)

45. **As to** claim **34** (Original) (incorporating the rejection in claim 32), please refer to claim **24** above, accordingly.

46. **As to claim 37** (Original) (incorporating the rejection in claim 32), please refer to claim **27** above, accordingly.

47. **As to claim 41** (Original) (incorporating the rejection in claim 32), please refer to claim **31** above, accordingly.

48. **As to claim 23** (Original) (incorporating the rejection in claim 22), Intel discloses a method wherein the cross-thread event is selected from a group consisting of a signal event, a wait event, a suspend event, a resume event, and a block event (e.g., Sec. 2 -Overview of the Critical Path Analysis, 2nd Para, 3rd Para;

Sec. 3.1 -Finding the critical path in the optimal case, 3rd Para; Sec. 3.1 - Finding the critical path in the optimal case, 4th Para. 5th Para; Sec. 3.3 - Finding the critical path with CPU constraints, 3rd Para, 4th Para)

49. Claim 23 is rejected under 35 U.S.C. 103(a) as being unpatentable over Hall in view of Hollingsworth-2 and Broberg and further in view of Intel.

50. **As to claim 23** (Original) (incorporating the rejection in claim 22), Broberg discloses a method wherein the cross-thread event is selected from a group consisting of a signal event, a wait event, a suspend event, a resume event, and a block event (e.g., Sec. 2 -Overview of the Critical Path Analysis, 2nd Para, 3rd Para; Sec. 3.1 - Finding the critical path in the optimal case, 3rd Para; Sec. 3.1 - Finding the critical path in the optimal case, 4th Para. 5th Para; Sec. 3.3 - Finding the critical path with CPU constraints, 3rd Para, 4th Para).

Further, Hall, Hollingsworth-2, and Broberg do not explicitly disclose a method wherein the cross-thread event is selected from a group consisting of a fork event, an entry event.

However, in an analogous art of *Performance Optimization using Critical Path Analysis in Multithreaded Programs on Multiprocessors*, Intel discloses a method wherein the cross-thread event is selected from a group consisting of a fork event, an entry event (e.g., P. 39, Right-Col, 1st Para; P. 38, Sec. of Multi-Entry Threading)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Intel into the Hall-Hollingsworth-2-Broberg's system to further provide a method wherein the cross-thread event is selected from a group consisting of a fork event, an entry event in Hall-Hollingsworth-2-Broberg system.

The motivation is that it would further enhance the Hall-Hollingsworth-2-Broberg's system by taking, advancing and/or incorporating Intel's system which offers significant advantages that Intel's Hyper-Threading Technology delivers two logical processors that can execute different tasks simultaneously using shared hardware resources; and, speculative pre-computation, a technique that improves the latency of single-threaded applications by utilizing idle multithreading hardware resources to perform long-range data pre-fetches as once suggested by Intel (e.g., P. 2, 2nd Para, 3rd Para)

Response to Arguments

51. Applicant's arguments filed on January 25, 2008 have been fully considered but they are not persuasive.

In the remarks, Applicant argues that, for examples:

(A.1) The combination Hall and Hollingsworth does not teach or suggest "monitoring information exchanged between a processing unit and first and second threads executed by the processing unit and determining a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree" (recited in REMARKS, page 8, second paragraph; page 10, first paragraph)

Examiner's response:

(R.1) Hall teaches "... call path profiling to handle multi-threaded server-style Java applications through the definition of two views: the Thread View, which profiles costs incurred by instances of single thread class, and the Server View, which **profiles CPU costs of all threads** ..." (e.g., Section 1 – Introduction, 1st Bullet) and "The **Thread View profiles** report (a) time spent when actually running, (b) time spent sharing **the CPU** with **equal priority threads**, (c) time spent waiting while **higher priority threads** are **running**, (d) time spent waiting for a monitor, and (e) idle time waiting for other **types of events**" (e.g., Sec. 1 – Introduction, the third Bullet)

Further, Hollingsworth (Hollingsworth-2) discloses "The **critical path** of a parallel program is **the longest path through the PAG (Program Activity Graph)**" (e.g., Sec. 2.1 – Formal Definition of Critical Path, first and second paragraphs) and "... The diagonal arcs show the communication events between processes and the dashed line **shows the critical path** through the program's execution ..." (e.g., Figure 1 – A program activity graph and calculating its critical path; Section 2 – Critical Path, first paragraph)

Conclusion

52. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Ben C Wang/

Examiner, Art Unit 2192

/Eric B. Kiss/
Eric B. Kiss
Primary Examiner, Art Unit 2192